ENGR 418 PROJECT REPORT

School of Engineering
Faculty of Applied Science
University of British Columbia

Project Title: Predicting Lego Piece Shapes using Linear Regression
Group No.: 12
Members: Drayton Monkman, Spencer Szabo
Date: November 16, 2021

Introduction

The purpose of this project is to develop a machine learning algorithm using python to train and test a multiclass classifier. Three classes of images are provided in two separate datasets: "training" and "testing". Our objectives are to use relevant machine learning techniques and python modules to properly train weights based on the "training" data set. Using these weights, we are to accurately classify the different types of images from the "testing" dataset, and present our results in the form of both a confusion matrix and accuracy indicator.

Theory

This project implements the use of a multiclass classifier. As its name suggests, a multiclass classifier is used to separate and classify multiple classes from datasets. To do this, the classifier must first be trained with a dataset similar to the dataset it will be classifying. There are two approaches to creating a multiclass classifier: One-versus-all, and multiclass softmax classification [1]. For our purposes we will continue with the one-versus-all approach.

The one-versus-all approach trains C two-class classifiers with each C classifier trained to distinguish each class from the rest of the data [1]. Here, each two-class classifier is learned using logistic regression, but they can be learned using other approaches as well. Through logistic regression, each data point will be assigned a label that recognizes if it is part of a certain class, or if it is not. Once the labels have been defined for each C two-class classifier, a vector that separates the labels is generated that represents the given weight for that class. Once all the weight vectors have been determined, the fusion rule (see appendix A) is used to combine the learned classifiers to make final assignments [1].

These methods are all implemented in the following algorithms using the python module Scikit-learn.

Algorithm

There are two algorithms utilized for this project, the training algorithm and the testing algorithm. Although similar, they have subtle differences. First of all, we have the training algorithm, implemented in the "TrainLinearModel" function, which works as described by the following pseudocode:

1) Load a list of training data from the 'Lego_dataset_1/training' directory
2) Iterate through each image to perform the following operations:
    a. Load the training image
    b. Crop the image (centered at the center of the original image)
    c. Resize to LxL pixels (choose L as a parameter)
    d. Vectorize the image
    e. Determine true class based on filenames (for training purposes only)
3) Train a logistic regression model using sklearn.linear_model.LogisticRegression
4) Return the trained model

After completion of model training, our training function will then test the model against the training data and print performance metrics, namely a confusion matrix and an accuracy score. Presumably this will be a perfect or near perfect result, although this is not guaranteed specifically if L is too small.

Secondly we have the testing algorithm, implemented in the "test_function" function, which works as described by the following pseudocode:

1) Iterating through each image to perform the following operations:
    a. Loading the testing image
    b. Cropping the image (centered at the center of the original image)
    c. Resizing to LxL pixels (L depends on the provided model)
    d. Vectorizing the testing image
    e. Predicting the class of image using the given model
    f. Determining true class based on filenames (optional, for testing accuracy only)
2) Compile resultant data for creation of classification report

The results of this algorithm are then displayed using a classification report and an accuracy score. With a well trained model, the accuracy score should be high, though likely not as high as when testing against the training data.

The primary parameter with regards to performance of our model is L, which represents both the side length of our coefficient matrix X as well as the side length of all square-cropped and re-scaled images. Although larger values of L may provide more accurate results, the training and calculation times increase exponentially with L, as does the risk of overfitting.

Given that we had a dataset of 54 training images split among 3 classes, we opted to choose L = 10 for our model. We felt that this value was large enough to avoid excessive loss of detail, but also small enough to avoid overfitting to the training data.

Results and Discussion

Overall our model performed well with the given value of L = 10.

```
Confusion matrix:
col_0  0.0  1.0  2.0
row_0
0.0     18    0    0
1.0      0   18    0
2.0      0    0   18
Accuracy Score: 1.0
```

Figure 1: Confusion Matrix and Accuracy Score of Model vs Training Data for L = 10

Figure 2: Visualization of Model Weights for L = 10

As shown in Figure 1, all images were correctly classified by our model resulting in a perfect accuracy score of 1.0. This tells us that training images are not losing too much data from downscaling. However, we cannot reasonably determine if our model is overfit when testing against the training data. To determine the true predictive accuracy of our model, we need to test against data not yet seen by the model.

Figure 2 shows us a visualization of the weights for our model, and helps us to understand what the model is looking for when making a prediction. Darker pixels mean more weight on the segment, and lighter means less weight. If the test image is dark in areas of high weight of one of the three shape predictors, it is more likely to be classified as that shape. For a circle, our model increases weights for the center of the image and reduces weight around the edges of the piece. For a rectangle, the model looks for distinct outcroppings on the top and bottom of the piece, while reducing weight everywhere else. For the square, our model looks more heavily at the right and left edges of the piece. Also note that, although our weight images are very pixelated, being more visually obvious to the human eye does not automatically make the model better, and inversely could be a sign of overfitting.

```
Confusion matrix:
col_0  0.0  1.0  2.0
row_0
0.0     17    0    1
1.0      0   16    2
2.0      1    0   17
Accuracy Score: 0.9259259259259259
```

Figure 3: Confusion Matrix and Accuracy Score Model vs Testing Data for L = 10

As shown in Figure 3, our model performed reasonably well against the testing data receiving an accuracy score of 0.926. Out of 54 test images, only 4 were misclassified. One circle (class 0) was misclassified as a square (class 2), one square (class 2) was misclassified as a circle (class 0), and two rectangles (class 1) were misclassified as squares (class 2). We can take a closer look at these cases and attempt to infer why they were misclassified.

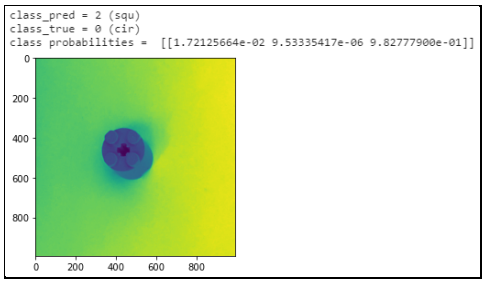

Figure 4: Misclassification 1 of Model vs Testing Data for L = 10

In Figure 4 we see a circle (class 0) being misclassified as a square (class 2) with a confidence of 0.983. It is not good for a model to be confidently wrong as it is in this case. When analyzing the image closer, we can see that the picture is taken at an odd angle, possibly being the main cause of this misclassification.

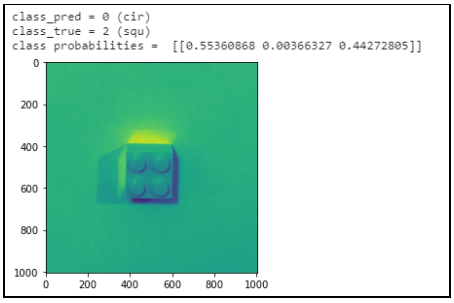

Figure 5: Misclassification 2 of Model vs Testing Data for L = 10

In Figure 5 we see a square (class 2) being misclassified as a circle (class 0) with a confidence of 0.554. This was close to being classified correctly, with the square confidence being 0.443, however the low contrast between the piece and the background is likely the cause of the error.

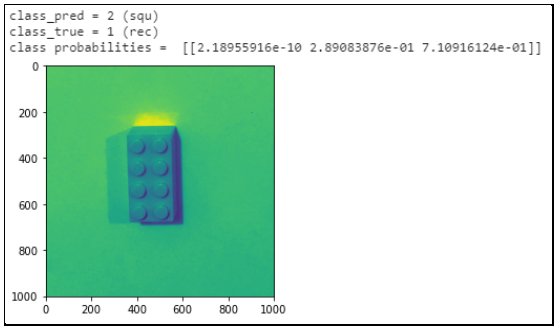

Figure 6: Misclassification 3 of Model vs Testing Data for L = 10

In Figure 6 we see a rectangle (class 1) being misclassified as a square (class 2) with a confidence of 0.711. This may be due to the dark shadow along the right edge of the piece, which as mentioned above is a place of emphasis for square classifications, in addition to the overall low contrast against the background.

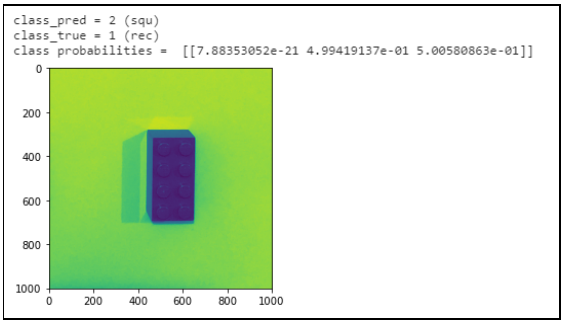

Figure 7: Misclassification 4 of Model vs Testing Data for L = 10

In Figure 7 we see another rectangle (class 1) being misclassified as a square (class 2), this time with a confidence of 0.501 despite high contrast. This was very close to being classified correctly, with the rectangle confidence being 0.499. The cause of this error could possibly be attributed to the piece being off center to the right, leading the image to miss some highly weight sections desired for rectangle classification.

Conclusions

In summary, we created a model that was capable of successfully predicting lego shapes in testing images with an accuracy of 0.926. The used methods were overall successful but not without room for improvement. In some cases the model was misled by imperfections in the photography. In others they may have been misled by the low contrast of the piece against the backdrop. Without resorting to expanding the training dataset, we could have potentially improved our model by utilizing multiclass classified logistic regression instead of OvA, or even opted for a different linear model entirely. Ultimately, the best way to improve our model would be to use a larger training dataset, increasing our model's awareness of real world imperfections, and allowing us to further increase L with less fear of overfitting.

Appendices

Appendix A: Fusion Rule

$$y = \underset{j=1,\ldots,C}{\operatorname{argmax}}\ b_j + \mathbf{x}^T \mathbf{w}_j.$$

Appendix B: Software Used
Microsoft Windows 10, Jupyterlab, Python 3.8.5

Appendix C: Proprietary Python Packages Used
Scikit Learn, pandas, matplotlib, NumPy, Python Imaging Library (PIL)

References

1. J. Watt, R. Borhani, and A. K. Katsaggelos, *Machine Learning Refined: Foundations, Algorithms, and Applications*. Cambridge: Cambridge University Press, 2016.